STAKECUBE: Combining Sharding and Proof-of-Stake to build Fork-free Secure Permissionless Distributed Ledgers

Antoine Durand, Emmanuelle Anceaume, Romaric Ludinard

▶ **To cite this version:**

Antoine Durand, Emmanuelle Anceaume, Romaric Ludinard. STAKECUBE: Combining Sharding and Proof-of-Stake to build Fork-free Secure Permissionless Distributed Ledgers. NETYS 2019: International conference on networked systems, Jun 2019, Marrakesh, Morocco. pp.148-165, 10.1007/978-3-030-31277-0_10 . hal-02078072

HAL Id: hal-02078072
https://hal.science/hal-02078072

Submitted on 25 Mar 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

STAKECUBE: Combining Sharding and Proof-of-Stake to build Fork-free Secure Permissionless Distributed Ledgers

Antoine Durand[1], Emmanuelle Anceaume[2], and Romaric Ludinard[3]

[1] IRT SystemX, Paris-Saclay, France antoine.durand@irt-systemx.fr
[2] CNRS/Université Rennes, France emmanuelle.anceaume@irisa.fr
[3] IMT Atlantique, France romaric.ludinard@imt-atlantique.fr

Abstract. Our work focuses on the design of a scalable permissionless blockchain in the proof-of-stake setting. In particular, we use a distributed hash table as a building block to set up randomized shards, and then leverage the sharded architecture to validate blocks in an efficient manner. We combine verifiable Byzantine agreements run by shards of stakeholders and a block validation protocol to guarantee that forks occur with negligible probability. We impose induced churn to make shards robust to eclipse attacks, and we rely on the UTXO coin model to guarantee that any stakeholder action is securely verifiable by anyone. Our protocol works against adaptive adversary, and makes no synchrony assumption beyond what is required for the byzantine agreement.

Keywords: Blockchain · Proof-of-Stake · Distributed Hash Table · Sharding

1 Introduction

Permissionless blockchains, also called distributed ledgers, initially appeared as the technological solution for the deployment of the Bitcoin digital cryptocurrency and payment system [23]. Permissionless blockchains aim at achieving the impressive result of being a persistent, distributed, consistent and continuously growing log of transactions, publicly auditable and writable by anyone. Despite the openness of the environment and thus the inescapable presence of malicious behaviors, security and consistency of permissionless blockchains do not demand the presence of a trusted third party.

This is a real achievement, which mainly results from the tight combination of two ingredients: a randomized election of the next block of transactions to be appended to the blockchain and a short latency broadcast primitive. While the latter one relies on the properties of peer-to-peer networks, the former one has so far been commonly implemented by solving proof-of-work (PoW), a cryptographic puzzle that is provably secure against a large proportion of participants that may wish to disrupt the system, and allows to keep the rate at which blocks are created parametrizable and independent of the size of the system. This second aspect is important to guarantee that the ratio between the message transmission delay and the block time interval remains low enough whatever the system activity, guaranteeing accordingly an easy management of conflicting blocks, if any.

Unfortunately, resilience of PoW-based solutions fundamentally relies on the massive use of computational resources, which is a real issue today. Lot

of investigations have been devoted to find a secure alternative to PoW, but most of them either rely on the intensive use of a large quantity of physical resources (*e.g.*, proof-of-space [5], proof-of-space/time [22]) or makes compromises in their trust assumptions (*e.g.* proof-of-elapsed-time [18], delegated proof-of-stake [14]). In contrast, solutions based on proof-of-stake (PoS) seem to be a quite promising way to build secure and permissionless blockchains. Indeed, proof-of-stake rely on a limited but abstract resource, the crypto-currency, in such a way that the probability for a participant to create the next block of the blockchain is generally proportional to the fraction of currency owned by this participant. It is an elegant alternative in the sense that all the information needed to verify the legitimacy of a stakeholder to create a block (i.e., crypto-currency possession) is already stored in the blockchain. Finally, by being a sustainable alternative (creating a block requires a few number of operations), scalability concerns, exhibited by PoW-based solutions, should be a priori more tractable.

An important condition for a PoS-blockchain to be secure is randomness. The creator of the next block must be truly random, and the source of randomness must not be biaised by any adversarial strategy. So far, this has been achieved by two main approaches: chain-based consensus and block-wise Byzantine agreement with respectively Ourobouros [8] and Algorand [16] as main representatives. In the former approach, a snapshot of the current users' status is periodically taken, from which the the next sequence of leaders is computed. In the latter one, a Byzantine agreement per block, relying on the properties of verifiable random cryptographic schemes, is achieved. High robustness against adaptive adversarial strategies results from the dynamic participation of thousands of users, each one participating for a single step of the algorithm.

In this paper we present a new blockchain protocol called StakeCube which aims at improving scalability of the block-wise Byzantine agreement approach by combining sharding techniques, users presence and stake transfer to operate in a PoS setting. The key idea of StakeCube is to organise users (*i.e.* stakeholders) into shards— such that the number of shards scales sub-linearly with the total number of active UTXOs— and within each shard, to randomly choose a constant size committee in charge of executing the distributed algorithms that contribute to the creation of blocks. Each block at height h in the blockchain is by design unique (no fork), and once a block is accepted in the blockchain, the next one is created by a sub-committee of shards whose selection depends on the randomness from the last accepted block.

To make such a solution correct in presence of a Byzantine adversary, we guarantee that the adversary cannot predict the shards in which users will sit, and that the sojourn time of users in their shard is limited. Doing so is an effective way to protect the system against eclipse attacks [2,6]. We introduce the notion of unpredictable and perishable users' credentials. Then to cope with this induced churn, shards' views are updated, signed and installed once, and this occurs right before the acceptation of a new block. Finally, the creation of blocks is efficiently handled by an agreement among a verifiable sub-committee of shards. We might expect that solely relying on stakeholders (*i.e.*, owners of the coins of the crypto-currency system) to the secure construction of the blockchain makes sense due to their incentive to be fully involved in the blockchain governance, rather than delegating it to powerful miners. However the analysis against rational players is left as future work.

The remaining of the paper organised as follows. Section 2 presents related work, Section 3 details our model and assumptions while Section 4 formalises the addressed problem. Section 5 describes an high-level view of the required building blocks of StakeCube while Section 6 presents the design principles of the proposed solution. A security analysis is provided in Section 7 before concluding in Section 8.

2 Related work

Omniledger [20] is the closest work to ours. It is a PoS-compatible, sharded, distributed ledger, resilient against a weakly dynamic adversary that corrupts up to $\frac{1}{4}$ of participants. In contrast to our approach, Omniledger assumes a strongly synchronous setting, and each shard maintains its own ledger and, global synchronisation of transactions is achieved through an atomic commit protocol tailored to their usage. Ouroboros [19], representative of the chain-based approach, is a synchronous PoS protocol resilient against a weakly dynamic adversary that owns $1/2-\varepsilon$ of stake. Moreover, Ouroboros has been recently improved to work in the partially synchronous setting against a dynamic adversary [8,7], but keeping the same design principles as the original one. In Ouroboros, a unique leader is elected at each round to broadcast its block which contrasts with our sharded approach were the block creation process is distributed. Snow White [13] is a synchronous PoS protocol resilient against a weakly dynamic adversary that owns $1/2$ of the *active* stake. This protocol also relies on a leader election. Algorand [16], is a representative of the blockwise Byzantine agreement approach. It provides a distributed ledger against an strongly adaptive adversary without assuming strong synchrony assumptions. However, by its design, agreement for each block of the blockchain is achieved by involving a very large number of stakeholders (*i.e.* several thousands) so that each one needs to effectively participate only for one exchange of messages.

3 Model

We assume a large, finite set of users whose composition may change over time. Users do not have synchronized clocks, but their individual clocks drift at the same rate. Users communicate by propagating messages within the system. The delivery of network messages is at the discretion of the adversary, but subject to synchrony assumptions. Our construction in itself makes no synchrony assumption except for what is required for the Byzantine resilient building blocks. Since our construction uses multiple building blocks, synchrony assumptions may be changed if they are instantiated differently than suggested. Users have access to basic cryptographic functions, including a cryptographic hash function $\mathfrak{h}$, and a CPA-secure signature scheme. Function $\mathfrak{h}$ is modeled as a random oracle.

Users own some minimal amount of stake (*i.e.* money), which gives them the right to participate to StakeCube. We adopt (a simplified version of) what is commonly known as the Bitcoin Unspent Transaction Output (UTXO) model. An UTXO can be roughly seen as a user's account credited by some stake. An UTXO is uniquely characterized by a public key pk_i and its associated amount of stake s_i. Each public key is related to the digital signature schema Σ with the uniqueness property, which allows stakeholders to use the public keys (or

a hash thereof) of their UTXOs as a reference to them, as demonstrated in the "Public Keys as Identities principle" of Chaum [10]. Note that the number of users evolves according to the UTXO set. At any time, a user can own multiple UTXOs. UTXOs can be debited only once, and once debited, an UTXO does not exist anymore. To simplify discussion, transactions outputs does not contains $\mathfrak{h}(pk_i)$ but directly pk_i.

Threat model: A weakly adaptive adversary We assume the presence of Byzantine (*i.e.* malicious) users which controls up to $\mu < 1/3 - \varepsilon$ of the total amount of stake currently available in the system. Here, ε quantifies the gain in the effective adversarial power, related to the security parameter. This model, named the "Stake Threshold Adversary" by Abraham and Malkhi [1], is an alternative to the common Threshold Adversary Model, which bounds the total number of parties the adversary controls relative to the total population of the system, and an extension (or modification) of the Computational Threshold Adversary introduced by Bitcoin, which bounds the proportion of the computational power owned by parties. Byzantine users can deviate from the protocol. They are modeled by an adversary. The adversary can perfectly coordinates all the malicious users. It can learn the messages sent by honest users (*i.e.* non malicious users), delay them, and then chooses messages sent by malicious ones. Further the adversary is weakly adaptive: it can select at any time which users to corrupt in replacement of corrupted ones (*i.e.* corruptions are "moving"), however a corruption becomes effective T blocks after the adversary has selected the user to be corrupted. The adversary is computationally bounded so that it can neither forge honest nodes' signatures nor break the hash function and the signature scheme. Finally, we assume that all users (honest and malicious) share an initial knowledge that we call *genesis block* which contains an initial arbitrary UTXO set.

4 The addressed problem

StakeCube aims at allowing any honest user i to locally maintain a sequence of blocks $B_0^i, B_1^i, \ldots, B_h^i$, where h represents the index (or the height) of the block in the sequence. This sequence is i's local copy of the distributed ledger, and satisfies both Safety and Liveness properties. In addition, the orchestration of the shards allows StakeCube to satisfy both Scalability and Efficiency properties. StakeCube is parametrized with an arbitrary security parameter κ, so that all its properties are guaranteed with probability at least $1 - e^{-O(\kappa)}$.

Property 1 (Safety). If honest user i accepts a block B_h^i at height h in its copy of the ledger then, for any honest user j that accepts a block at height h in its copy ledger, $B_h^j = B_h^i$.

Property 2 (Liveness). If a honest user submits transaction tx, then eventually tx appears in a block accepted in the copy of all honest users.

In StakeCube, participation of honest users is conditional to the possession of UTXOs. Participation is voluntary: Any honest user can join a shard (determined by the protocol), whenever she wishes, with the objective of eventually being involved in the Byzantine resilient protocols executed in this shard.

Participation is temporary: The sojourn time of an honest user in a shard is defined by the time it takes for StakeCube to create T blocks. Once she leaves, she can participate again by joining another shard, and does so until she spends her UTXO. As users may own multiple UTXOs, they can simultaneously and verifiably sit in different shards. In the following, a user that issued a join request with its current credential is called an *active* user. StakeCube satisfies Scalability and Efficiency properties. This is achieved by the properties of our block creation process. Adding a new block takes two byzantine fault tolerant protocols to be run in parallel within each shard, one network wide diffusion by each shard, one inter-shard byzantine agreement, and finally one broadcast for the block.

Property 3 (Scalability). All Byzantine fault tolerant protocols we rely on have an (overall) $O(n^3)$ communication complexity. However in StakeCube these protocols are executed by committees whose size is small and fixed. Because the number of shard is $O(\sqrt{N})$, the overall communication cost is $O(NC_1^3 + C_2^3)$, with C_1 and C_2 some constants depending on κ. Thus, each participant's average communication cost is sublinear.

Property 4 (Efficiency). All Byzantine fault tolerant protocols we rely on uses a constant number of rounds. Thus adding a new block also takes a constant number of rounds. Because a transaction, once diffused, will be included in the next block and that blocks are final, it takes at most two blocks to include a new transaction.

5 A set of ingredients

To solve the addressed problem, StakeCube relies on the orchestration of the following ingredients.

Cryptographic primitives Digital signature together with random hash functions allow the implementation of verifiable random functions (VRF) [21]. In a VRF, a secret key sk allows the evaluation of hash function $\mathfrak{h}$ on input x as well as the computation of a non-interactive proof that shows that the sk is the only one that can compute y. Verification of the proof is done with respect to the public key pk only. The proofs must remain sound even when pk was computed maliciously and $\mathfrak{h}(sk,x)$ must remain pseudorandom even when an adversary can query values of $\mathfrak{h}$ and proofs for them for any input value x'.

Byzantine Vector consensus A vector consensus protocol [12] is a byzantine resilient protocol where n participants agree on a vector representing the input value of each participant. Validity condition states that in presence of $f \leq \lfloor (n-1)/3 \rfloor$ byzantine nodes, the vector contains at least $f+1$ non-null values, and for each non-null value $v_i \neq \perp, 1 \leq i \leq n$, this value was initially proposed by participant i.

Random beacon A Random beacon is a service that provides a public source of randomness. It was first proposed by Rabin [24] in the context of contract signing. In our case, we need the random beacon to be emulated by a distributed protocol without trusted third parties. That is, a protocol that satisfies the following security properties:

1. *Guaranteed output delivery.* All honest participants eventually output a value.

2. *Unpredictable.* Any adversary's ability to predict any information about the beacon prior to it being published is negligible.
3. *Unbiased.* For all adversarial strategies, the output is statistically close to a uniformly random string.
4. *Publicly verifiable.* The protocol also produces a proof that can be verified by third parties to be convinced that a beacon is indeed the output of the protocol.

Suitable instantiations for the distributed setting includes SCRAPE [9] and RandHerd [25]. In the following we denote by μ_{core} the minimum of the fractional resiliency of the vector consensus and random beacon protocols.

Verifiable Byzantine agreement We use a verifiable byzantine agreement in order to agree on the next signed block despite corrupted shards. Our main requirement for this algorithm is to be optimistic, *i.e.* efficient in the absence of faults. Indeed, the analysis in Section 7 shows that the probability for a shard to be corrupted exponentially decreases with shards core size. Any verifiable Byzantine algorithm satisfying our assumptions can be used. We rely on the solution proposed by Shen et al [11] since it is leader-based, efficient and tolerant to temporary partitions. The fractional resiliency of this protocol is noted $\mu_{corrupted}$.

Distributed Hash Table (DHT). Distributed hash tables (DHTs) build their topology according to structured graphs, and for most of them, the following principles hold: each node of the system has an assigned identifier, and the identifier space, *e.g.*, the set of 256-bit strings, is partitioned among all the nodes of the system. Nodes self-organize within the graph according to a distance function based on the identifier space.

Sharded DHT. The notion of Sharded DHT is similar to a regular DHT, except that each vertex of the DHT is a set of nodes instead of a single node. That is, nodes gather together into shards, and shards self-organize into a DHT graph topology. Sharded DHTs can be made robust to adversarial strategies as achieved in SChord [15], and PeerCube [3], and robust to high churn as achieved in PeerCube [3] by running Byzantine tolerant algorithms within each shard. For these reasons, we rely on PeerCube architecture, while weakening its model by removing the assumption of a global trusted party supplying verifiable random identifier, and by removing the assumption of a static adversary. For self-containment reasons, we now recall the main design features of PeerCube. Briefly, this is DHT that conforms to an hypercube. Each vertex (*i.e.* shard) of the hypercube is dynamically formed by gathering nodes that are logically close to each other according to a distance function applied on the identifier space. Shards are built in way to ensure that the respective common prefix of their members is never a prefix of one-another. This guarantee that each shard has a unique common prefix, that in turn serves as a shard's *label*. The shard's label characterizes the position of the shard in the overall hypercubic topology, as in a regular DHT. Shards size is upper and lower bounded. Whenever the size of shard $\mathcal{S}$ exceeds a given value $s_{\max}$, $\mathcal{S}$ splits into shards of higher degree, and whenever the size of $\mathcal{S}$ falls under a given size of $s_{\min}$ nodes, $\mathcal{S}$ merges with another shard into a single new shard of lower degree. Each shard self-organizes into two sets, the core set and the spare set. The core set is a fixed-size random subset of the whole shard. It is responsible for running the Byzantine agreement protocols, to guarantee that each shard behaves as a single and correct entity (by for example forwarding all the

join and lookup requests to their destination), despite malicious participants [4]. Members of the spare set merely keep track of shard state. Joining the core set only happens when some existing core member leaves, in which case the new core set member is randomly elected among the spare set. By doing this, nodes joining the system weakly impact the topology of the hypercube [3].

6 Design Principles of StakeCube

StakeCube allows the creation of a permissionless distributed ledger in a PoS setting. The key idea of StakeCube is to organise users (*i.e.* stakeholders) into shards— such that the number of shards scales sub-linearly with the total number of active UTXOs— and within each shard, to randomly choose a constant size committee in charge if executing the distributed algorithms that contribute to the creation of blocks. Each block at height h in the blockchain is unique, and once a block is accepted in the blockchain, the next one is created by a sub-committee of shards whose selection depends on the random seed of the last accepted block.

To be able to tolerate the presence of a Byzantine adversary, we must guarantee that the adversary cannot predict the shards in which users will sit, and that the sojourn time of users in their shard is limited. To achieve this, we introduce the notion of unpredictable and perishable users' credentials in Section 6.1. Then to cope with this induced churn, we show how to update, sign a install the shards' views in Section 6.2. This process occurs right before the acceptation of a new block. Finally, the creation of blocks is efficiently handled by an agreement among a verifiable sub-committee of shards (see Section 6.3).

6.1 Unpredictable and perishable users' credentials

As described in Section 5, Peercube critically relies on a (global) trusted party supplying verifiable random identifiers to nodes. In this section, we detail how to construct those in our decentralized setting, using the already known public keys and some common randomness present in each block.

For each unspent public key, *i.e.* for each UTXO, owned by a user, a sequence of unpredictable and perishable credentials are tightly assigned to her. Validity of a credential spans T blocks, with T some positive integer. The credential σ assigned to user i for its UTXO (pk_i, sk_i) is computed as follows. Let B_{h_0} be the block at height h_0 of the blockchain such that pk_i was created in B_{h_0}, *i.e.*, it exists a transaction in B_{h_0} such that pk_i appears in the output list of that transaction. For any blockchain height $h \geq h_0 + T$, such that UTXO (pk_i, sk_i) still exists when B_h is accepted in the blockchain,

$$\sigma_{pk_i}(h) := H(pk_i || B_{h'}.\rho), \quad \text{where} \quad h' := h_0 + \lfloor \frac{h - h_0}{T} \rfloor T, \tag{1}$$

with $B_{h'}.\rho$ a random number whose computation is detailed in Section 6.3. Suppose that i's UTXO (pk_i, sk_i) is created in block B_h. Then by Relation 1, i's first credential for UTXO (pk_i, sk_i) is computed based on the content of block B_{h+T} and perishes at block B_{h+2T}. Then, i's second credential for (pk_i, sk_i) is computed based on the content of block B_{h+2T} and perishes at block B_{h+3T}, and so on until i spends (pk_i, sk_i).

User i's credential uniquely characterizes the shard to which user i is allowed to sit, and this shard is the one whose label prefixes i's current credential

$\sigma_{pk_i}(h)$. By the non-inclusion property of PeerCube [3], there does not exist a shard whose label is the prefix of another shard, and thus, there is a unique shard whose label prefixes credential $\sigma_{pk_i}(h)$. When her current credential expires, i leaves the shard she is in, and if she wants to continue to participate to StakeCube, joins a new shard based on her new credential.

There are a couple of details that should be noted.

1. User i does not need to participate in StakeCube for the entire life of her UTXO (pk_i, sk_i). She can join StakeCube (*i.e.* join a shard) at any time h under credential $\sigma_{pk_i}(h)$, however once a user joins her shard, she must stay online (and actively participates if she is a core member) until $\sigma_{pk_i}(h)$ expires. As a result, there exists no explicit leave request, it simply consists in not issuing a join request upon credential renewal. A consequence of this rule is that, in case user i participates under credential $\sigma_{pk_i}(h)$ and spends her UTXO (pk_i, sk_i) before $\sigma_{pk_i}(h)$ expires, then i continues to participate under $\sigma_{pk_i}(h)$ until $\sigma_{pk_i}(h)$ expires. Note that because a transaction only grants credentials after a delay, this rule does not allow a user to simultaneously own multiple credentials for the same stake. Note also that if i is disconnected for a small amount of time this does not jeopardized the safety of the shard only its liveness.
2. Recall that the adversary has a bounded fraction μ of *stake* in StakeCube. To defend StakeCube against Sybil attacks (*i.e.*, the fact that the adversary creates a considerable number of UTXOs with the objective of overpopulating each shard with malicious owners of those UTXOs), we require that each UTXO cannot be credited with more than M stake, with M some predefined constant. Consequently, by the fact that for any $h > 0$ one credential $\sigma(h)$ represents exactly one UTXO, there is a bound $\mu_{cred} > \mu$ on the fraction of malicious credentials in StakeCube, which is reached when all malicious UTXOs have 1 stake and all honest ones maximize their stake, *i.e.*, each honest UTXO has M stake. Note that UTXOs with M' stake, such that $M' > M$ may be handled by granting them $\lceil M'/M \lceil$ credentials, although we do not treat this case explicitly. Section 7 analyzes the distribution of malicious credentials among shards.

Regarding the behavior of the adversary, there are a couple of remarks to note.

1. At any time, the adversary might spend some selected UTXOs in order to create new ones and thus new credentials with the objective of targeting some shards. However, because of the initial T blocks delay required to obtain the first credential for an UTXO (see Relation 1), any newly created UTXO will give rise to a credential only after all existing credentials are renewed as well. Therefore, the adversary has no preferred strategy regarding transactions and forced renewal.
2. Each block B_h contains a random seed, denoted by $B_h.\rho$, which cannot, by construction, be neither biased nor predictable before the block is created (how such seeds are generated is detailed in Section 6.3), the adversary cannot determine nor influence the value of renewed credentials. Consequently, for any blockchain height $h \geq 0$ and for any pk_i, $\sigma_{pk_i}(h+T)$ is unpredictable while for any $0 \leq h' \leq h$, the sequence $(\sigma_{pk_i}(h'+T))_{0 \leq h' \leq h}$ is computable and verifiable from the blockchain.

6.2 Shard membership

As described above, during the period of time that elapses between the creation of an UTXO to its spending, the UTXO owner can participate to the blockchain construction by successively joining a series of shards. In practice this may give rise to a voluminous amount of join requests, which might be highly prejudicial to STAKECUBE's scalability and efficiency if each joining request led to the insertion of the newcomer in the core which run the distributed operations. Rather, by relying on PeerCube design (see Section 5), a newcomer joins the spare set of the shard as a candidate for being elected as a member of the core set of the shard, and participate then to the distributed algorithms. Management of the view composition, and election in the core set is the purpose of the remaining of the section.

View of a shard. The view of a shard $\mathcal{S}$ reflects the composition of both its core and spare sets, denoted respectively $\mathcal{S}_c$, $\mathcal{S}_s$. Update of the view is strongly correlated to blockchain events: any block appended to the blockchain is preceded, in each shard, by the update and the installation of the shard view. In the following, the view of shard $\mathcal{S}$ installed right before block B_h is appended to the blockchain is denoted by $view_{\mathcal{S}}(h)$. We have $view_{\mathcal{S}}(h) = (\mathcal{S}_c(h), \mathcal{S}_s(h))$, where $\mathcal{S}_c(h)$ (resp. $\mathcal{S}_c(h)$) represent the composition of $\mathcal{S}$'s core set (resp. spare set) at time h.

Update of the shard view. When a newcomer (*i.e.* a user under a valid credential) issues a request to join her shard $\mathcal{S}$, her request is propagated and broadcast to the members of $\mathcal{S}_c$. Core members i locally store the join request in their buffer b_i of pending requests. Note that expiration of credentials do not need to be locally memorized, prior to being handled by the view update algorithm, since by Relation 1, credentials can only expire when a new block is appended to the blockchain. Let $view_{\mathcal{S}}(h-1)$ be the current view of $\mathcal{S}$ when a (honest) core member $i \in \mathcal{S}_c(h-1)$ receives some valid block B_h (Section 6.3 details the creation of blocks). The following three steps are successively executed:

1. A Byzantine vector agreement protocol is run among $\mathcal{S}_c(h-1)$ members to decide on the set of newcomers: core members i propose their local buffer b_i, and the outcome of the protocol is a vector $v(h)$ of newcomers such that non-null values for honest core members i are equal to their buffer b_i. Each honest core member i replaces its local buffer b_i with the union of the users of the decided vector. We have $b_i = \cup_{b_j \in v(h), b_j \neq \perp} b_j$.
2. Each user $i \in \mathcal{S}_c(h-1)$ removes from b_i the set of $r_{\mathcal{S}}(h)$ of users whose credential expires with B_h. User i initializes a new spare set $S_s(h)$ with $b_i \cup S_s(h-1) \setminus r_{\mathcal{S}}(h)$, and orders $S_s(h)$.
3. Each user $i \in \mathcal{S}_c(h-1)$ initializes a new core set $S_c(h)$ with $S_c(h) = S_c(h-1) \setminus r_{\mathcal{S}}(h)$. If $\mathcal{S}_c(h-1) \cap r_{\mathcal{S}}(h) \neq \emptyset$, some previous core members $i \in \mathcal{S}_c(h-1)$ have credential that expire with B_h. As a consequence, an election among the users of $\mathcal{S}_s(h)$ is carried out for i's replacement, so as to keep $|\mathcal{S}_c(h)| = s_{\min}$. The core election works as follows:
   (a) A random beacon protocol is run among $S_c(h-1)$ members to decide on a common random seed ρ.
   (b) A pseudo-random number $PRG(\rho)$ generator is initialized with ρ as seed.

(c) $PRG(\rho)$ is used to draw a random number $j \in [\![1, |S_s(h)|]\!]$. The j-th member of $\mathcal{S}_s(h)$ is removed from $\mathcal{S}_s(h)$ and added to $\mathcal{S}_c(h)$. This process is repeated until $|\mathcal{S}_c(h)| = s_{\min}$.

Once these steps are completed, each core member j installs her new view $view^j_{\mathcal{S}}(h)$ with the new values of $S_c(h)$ and $S_s(h)$, signs it, and sends it to the spare members. Once a spare receives $\mu_{core} s_{\min} + 1$ signatures on the same view, it installs it. In the meantime, each core member j resets its buffer $b_j = \emptyset$.

Note that multiple join requests may lead a shard $\mathcal{S}$ to split into two shards, or, on the contrary, may lead two shards $\mathcal{S}'$ and $\mathcal{S}''$ to merge within a single one $\mathcal{S}$. The treatment of such topological changes are omitted in the above procedure for space reasons, but can be derived from the description that appears in [2].

To summarize, the shard membership procedure ensures that, for any shard $\mathcal{S}$ of StakeCube, all members of $\mathcal{S}$ install the same view $view_{\mathcal{S}}(h)$ before appending block B_h to their copy of the blockchain.

Diffusing views. Merely installing the new view for each shard is not sufficient. We need the other shards of StakeCube to maintain this knowledge to be able to verify any signed information exchanged during inter-shard communication (*e.g.* during the block proposal procedure). Therefore, whenever a new view $view_{\mathcal{S}}(h)$ is installed along with its $\mu_{core} s_{\min} + 1$ signatures, it is also broadcast to the whole network as a notification of the view update. Note that shards only store the last view $view_{\mathcal{S}'}(h)$ of any other shard $\mathcal{S}'$ and not its whole history.

6.3 Construction of the next block of the blockchain

In the following we propose a byzantine resilient cross-shard mechanism to agree on a unique valid block, despite the presence of at most f_{shard} corrupted shards (see section 7 for the computation of f_{shard}). Indeed, the presence of an adaptive adversary may compromise the safety of some shards by succeeding in having more than a proportion μ_{core} of malicious users sitting in the core of some shards. Although the probability of such event can be made arbitrarily low (see the analysis presented in Section 7), we must handle it. The presence of corrupted shards put us in the same situation as in a consensus protocol: given the same initial chain, any shard is able to create the next block, and the decision must be a unique block, despite malicious users lying or not responding. As will be now described, this is efficiently and robustly achieved by running a Verifiable Byzantine agreement among a subset of shards of StakeCube selected by relying on block's seed.

Reaching consensus on the next block. The process of creating a new block B_h starts right after B_{h-1} has been accepted (see Section 6.2). A committee of shards, denoted in the sequel by $\mathbb{C}$, is elected among the shards of StakeCube. The election of each of these shards relies on the seed of block B_{h-1}, derived from the random beacon protocol (see Section 5). Once elected, the committee executes a verifiable Byzantine agreement to decide on the unique block B_h to be appended to the blockchain. The main steps of this process are as follows:

1. All shards compute the elected committee $\mathbb{C}$, similarly to the core election procedure (see Section 6.2):

(a) Let $\mathbb{L}$ be the set of shards's label such that $view_{\mathcal{S}'}(h-1)$ has been received. $\mathbb{L}$ is then ordered through a canonical order.
(b) A pseudo-random number generator $PRG(B_{h-1}.\rho)$ is initialized, where $B_{h-1}.\rho$ is the seed of the last block B_{h-1}.
(c) $PRG(B_{h-1}.\rho)$ is used to draw a random number $j \in [\![1, |\mathbb{L}|]\!]$. The j-th member of $\mathbb{L}$ is removed from $\mathbb{L}$ and added to $\mathbb{C}$ (initially initialized to $\emptyset$). This process is repeated until $\mathbb{C}$ contains $s_{\mathbb{C}}$ shards, with $s_{\mathbb{C}} = (f_{shard}/\mu_{corrupted}) + 1$. Recall that f_{shard} is the maximal number of corrupted shards in StakeCube (whose computation is presented in Section 7), and $\mu_{corrupted}$ is the fraction of malicious nodes tolerated by the verifiable Byzantine Agreement protocol (see Section 5).

2. Then, committee $\mathbb{C}$ members run the verifiable Byzantine Agreement protocol, with their proposed block B_h as input (the next paragraph describes the construction of the proposed block). Finally the decision is a block b_h signed by $2f_{shard}+1$ shards.
3. Block b_h is broadcast in StakeCube and appended to StakeCube users' local copy of the blockchain.

Security remark: By $s_{\mathbb{C}}$'s value, committee $\mathbb{C}$ cannot be corrupted, independently of the shards selected by the election. Committee $\mathbb{C}$ is still composed by randomly selected shards because it naturally spreads the load of creating a block, while preventing corrupted shards from trying to manipulate the election process to get in the committee and slow it down. Note that the random seed is already available and thus does not need to run a distributed random beacon.
Efficiency remark: We rely on a leader-based BA algorithm to benefit from its optimistic efficiency. Indeed, since f_{shard} can be made arbitrarily small (see Section 7), and committee $\mathbb{C}$ members are randomly selected, we expect the first leader to almost always be an honest shard.

Construction of the proposed block. We finally describe how each shard $\mathcal{S}$ of $\mathbb{C}$ constructs the block B_h to be proposed as input of the verifiable Byzantine agreement protocol. The construction results from an agreement among the core members of $\mathcal{S}$ on the content of the block and on the generation of the block's seed. Let $view_{\mathcal{S}}(h) = (\mathcal{S}_c(h), \mathcal{S}_s(h))$ be the current view of shard $\mathcal{S}$. The main steps of this creation process are as follows:

1. Each core member in $\mathcal{S}_c(h)$ proposes *(i)* its list of pending transactions and *(ii)* its VRF value seeded with $B_{h-1}.\rho$ together with the VRF proof, to the Byzantine Vector consensus protocol. The decision value is a vector of inputs values, such that non-null values for honest core members are equal to their list of pending transactions and their VRF value and VRF proof.
2. Construction of block B_h is then realized as follows
   - The hash of the previous block B_{h-1} is inserted in B_h's header.
   - The union of transactions from the decided vector defines B_h's body.
   - The hash of the concatenation of the VRF values of the decided vector defines the seed $B_h.\rho$ of B_h.
   - The list of VRF proofs of the decided vector is inserted in B_h's header as a proof of randomness for seed $B_h.\rho$.

The reason why we do not use the random beacon protocol is because it is supposed to be generated in a non corrupted set. Here we have different requirements:

– We want the seed to stay close to random even in the case of corrupted shard. This does come at the cost of giving the adversary a bounded number of choices for the seed.
– We do not mind the fact that the adversary aborts the computation of the seed, because a malicious shard can already decide to not create a block anyways.

7 Security Analysis

In this section we analyse the probability that some of the shards of StakeCube are corrupted, that is that their core set contain more than $\mu_{core} s_{\min}$ malicious users. In the following we denote by ν the fraction of corrupted shards. To conduct such an analysis, we examine a simplified scenario. We approximate the behavior of StakeCube by taking the amortized execution over one period of T blocks. That is, we study the corruption probability when all the shards are built and the cores are elected, once. In particular, this is equivalent to the case were all credentials are synchronously renewed at the same block say block B_h. Note that, for a fixed number of active users, the number of credential renewals, core election, and topological changes is statistically the same for every period of length T.

7.1 Corruption probability of a core during a period of T blocks

Let s be the size of shard $\mathcal{S}$, μ_{shard} be a bound on the ratio of malicious users within $\mathcal{S}$, and μ_{core} be the fractional resiliency of the vector agreement protocol and random beacon one, with $0 \leq \mu_{shard} < \mu_{core} \leq \mu$. We compute an upper bound on the probability that the fraction of malicious users in the core is higher than μ_{core} by the end of the period.

As seen in Section 6.2, the core set is elected by randomly taking $s_{\min}$ credentials from the shard $\mathcal{S}$, without replacement. Let Y the random variable equal to the number of malicious credentials within the core, *i.e.*, Y follows an hypergeometric distribution.

$$\forall k \in [\![0, s_{\min}]\!], \mathbb{P}[Y = k] = \frac{\binom{\lfloor s\mu_{shard} \rfloor}{k}\binom{\lfloor s(1-\mu_{shard}) \rfloor}{s_{\min}-k}}{\binom{s}{s_{\min}}}.$$

We are interested in deriving the probability that, after T blocks, *i.e.* after T core renewals, the core set $\mathcal{S}$ is corrupted. The core set corruption refers to the situation where the proportion of malicious credentials in the core exceeds μ_{core}. Applying the Hoeffding bound [17] on the previous relation leads to

$$\mathbb{P}[\frac{Y}{s_{\min}} \geq \mu_{core}] \leq e^{-2(\mu_{core}-\mu_{shard})^2 s_{\min}}.$$

Thus, assuming that the fraction of malicious users in a shard is below μ_{shard}, the corruption probability over T blocks exponentially decreases when $s_{\min}$ increases.

7.2 Distribution of malicious credentials among all shards

The above section assumes that the fraction of malicious users in all shards is below μ_{shard}. In this section we compute an upper bound on the probability that this assumption does not hold. To do so, we make simplification assumptions on how the shards are formed. First, we assume there are K shards of size S, *i.e.* $N := SK$ credentials in total. Second, we assume that the shards configuration relevant to the period results from the credentials filling at random all the shards. Recall that μ_{cred} is the overall ratio of malicious credentials.

Let us consider the random variable Y corresponding to the number of malicious credentials in a given shard. Let us consider a set $\{Y_1,\dots,Y_K\}$ of random variables identical to Y, and let $\mathbf{Y} = (Y_1,\dots,Y_k) \in \{0,S\}^K$ a vector made of these random variables. The vector $\mathbf{Y}$ represents the distribution of malicious credentials in StakeCube. Random variable $\mathbf{Y}$ follows a multivariate hypergeometric distribution: Each of the $N = SK$ credentials is assigned a shard, and we analyse the shard assignment of a random sample of size $N\mu_{cred}$. We define set I as the set of vectors representing StakeCube where $N\mu_{cred}$ credentials are malicious. We have:

$$\mathrm{I} = \{\mathbf{x} \in [0,S]^K \mid \sum_{i=1}^{K} x_i = N\mu_{cred}\} \quad \text{and} \quad \forall \mathbf{x} \in \mathrm{I}, \mathbb{P}[\mathbf{Y} = \mathbf{x}] = \binom{N}{N\mu_{cred}}^{-1} \prod_{i=1}^{K} \binom{S}{x_i}.$$

We are interested in computing the probability that a given shard $1 \leq j \leq K$ contains more than m malicious credentials. Let us define $\mathrm{I}_{m,j} = \{\mathbf{x} \in \mathrm{I} \mid x_j \geq m\}$. We have:

$$\begin{aligned}\mathbb{P}[\mathbf{Y} \in \mathrm{I}_{m,j}] &= \mathbb{P}[\mathbf{Y} \in I, Y_1 \geq 0, \dots, Y_j \geq m, \dots, Y_K \geq 0] \\ &= \sum_{k=m}^{S} \binom{S}{k} \binom{N}{N\mu_{cred}}^{-1} \prod_{1 \leq i \leq K, i \neq j} \binom{S}{x_i}\end{aligned}$$

Knowing that $\sum_{1 \leq i \leq K, i \neq j} x_i = N\mu_{cred} - k$ and $\sum_{1 \leq i \leq K, i \neq j} S = N - S$, we can apply Vandermonde's identity:

$$\forall j, \mathbb{P}[\mathbf{Y} \in \mathrm{I}_{m,j}] = \sum_{k=m}^{S} \binom{N}{N\mu_{cred}}^{-1} \binom{S}{k} \binom{N-S}{N\mu_{cred}-k}$$

We now get our result by applying first the (univariate) Hoeffding bound, and then the union bound.

$$\forall j, \mathbb{P}[\mathbf{Y} \in \mathrm{I}_{S\mu_{shard},j}] \leq e^{-2(\mu_{shard}-\mu_{cred})^2 S}$$

Thus the probability that at least one shard of the system contains more than $\mu_{shard}S$ malicious credentials is bounded by

$$\mathbb{P}[\mathbf{Y} \in \cup_{j=1}^{K} \mathrm{I}_{S\mu_{shard},j}] \leq Ke^{-2(\mu_{shard}-\mu_{cred})^2 S} = e^{-(2(\mu_{shard}-\mu_{cred})^2 S - \ln K)}.$$

As required, term $\cup_{j=1}^{K} \mathrm{I}_{S\mu_{shard},j}$ is the set of shards assignations to malicious credentials, such that at least one shard has a fraction greater than or equal to μ_{shard} of malicious credentials.

Moreover, due to the union bound, this upper bound also holds if the shards have different sizes and S is the minimum, hence, we can simply use $S = s_{\min}$. As for K, the worst case is reached when there is a maximal number of shards, *i.e.* $K = N/s_{\min}$.

7.3 Putting it all together

In the previous subsection we obtain exponentially decreasing bounds on the probability that at least one shard is corrupted, *i.e.*, proving security when the bound on the number of malicious shards f_{shard} is set to 0. We let for future work the generalization of this calculation with arbitrary values of f_{shard}, which would give us tighter parameters.

The adversary has a fraction μ of stake. Requiring each credential to be associated to at most M stake gives us the the following (worst case) ratio of malicious credentials, which is reached when all malicious UTXOs have 1 stake and all honest ones maximize their stake, *i.e.*, each honest UTXO has M stake. We then have

$$\mu_{cred} = \frac{1}{1+M^{-1}(\mu^{-1}-1)}.$$

Thus M should be as small as possible to decrease the adversary effective stake. However low values of M requires users to participate with multiple credentials in parallel, increasing the communication cost for individual users. Knowing μ and security parameter κ, the parameters μ_{shard} and $s_{\min}$ can be obtained by solving:

$$\mu_{shard} \leq \mu_{cred} + \sqrt{\frac{\kappa - \ln \frac{N}{s_{\min}}}{2s_{\min}}} \quad \text{and} \quad s_{\min} \geq \frac{\kappa}{2(\mu_{core} - \mu_{shard})^2}.$$

8 Conclusion & future work

In this paper we have presented StakeCube a new blockchain protocol which aims at improving scalability of the block-wise Byzantine agreement approach by combining sharding techniques, users presence and stake transfer to operate in a PoS setting. Each block at height h in the blockchain is by design unique (no fork), and once a block is accepted in the blockchain, the next one is created by a sub-committee of shards whose selection depends on the random seed of the last accepted block.

The next step is to take into account the stake associated with each credential as weights into both the core election and the election of the shard in charge of creating the next block. This will allow us to get rid of the $\mu_{cred} - \mu$ gain in adversarial power, while keeping the remaining of the security arguments similar.

References

1. Abraham, I., Malkhi, D.: The blockchain consensus layer and bft. Bulletin of the European Association for Theoretical Computer Science (123) (2017)
2. Anceaume, E., Sericola, B., Ludinard, R., Tronel, F.: Modeling and Evaluating Targeted Attacks in Large Scale Dynamic Systems. In: International Conference on Dependable Systems and Networks (DSN) (2011)
3. Anceaume, E., Ludinard, R., Ravoaja, A., Brasileiro, F.: PeerCube: A Hypercube-Based P2P Overlay Robust against Collusion and Churn. In: Proceedings of the IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO) (2008)
4. Anceaume, E., Ludinard, R., Sericola, B.: Performance evaluation of large-scale dynamic systems. ACM SIGMETRICS Performance Evaluation Review **39**(4) (2012)

5. Ateniese, G., Bonacina, I., Faonio, A., Galesi, N.: Proofs of Space: When Space Is of the Essence. In: Proceedings of the International Conference on Security and Cryptography for Networks (SCN) (2014)
6. Awerbuch, B., Scheideler, C.: Towards scalable and robust overay networks. In: Proceedings of the International Workshop on Peer-to-Peer Systems (IPTPS) (2007)
7. Badertscher, C., Gaži, P., Kiayias, A., Russell, A., Zikas, V.: Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In: Proc. of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS) (2018)
8. Bernardo, D., Gaži, P., Kiayias, A., Russell, A.: Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In: Proc. of International Conference on the Theory and Applications of Cryptographic (EUROCRYPT) (2018)
9. Cascudo, I., David, B.: SCRAPE: Scalable randomness attested by public entities. In: Proc. of the International Conference on Applied Cryptography and Network Security (ACNS) (2017)
10. Chaum, D.: Untraceable electronic mail, return addresses, and digital pseudonyms. Communications of the ACM **24**(2), 84–90 (1988)
11. Chen, J., Gorbunov, S., Micali, S., Vlachos, G.: Algorand agreement: Super Fast and Partition Resilient Byzantine Agreement. Tech. rep. (2018), https://eprint.iacr.org/2018/377
12. Correia, M., Neves, N.F., Veríssimo, P.: From consensus to atomic broadcast: Time-free Byzantine-resistant protocols without signatures. The Computer Journal **49**(1) (2006)
13. Daian, P., Pass, R., Shi, E.: Snow White: Provably Secure Proofs of Stake. Cryptology ePrint Archive, Report 2016/919 (2016), https://eprint.iacr.org/2016/919
14. EOS.IO: Technical white paper v2 (2019), https://github.com/EOSIO/Documentation/blob/master/TechnicalWhitePaper.md, accessed: 2019-03-10
15. Fiat, A., Saia, J., Young, M.: Making chord robust to byzantine attacks. In: Proceedings of the Annual European Symposium on Algorithms (AESA) (2005)
16. Gilad, Y., Hemo, R., Micali, S., Vlachos, G., Zeldovich, N.: Algorand: Scaling Byzantine Agreements for Cryptocurrencies. In: Proceedings of the 26th Symposium on Operating Systems Principles (SOSP) (2017)
17. Hoeffding, W.: Probability inequalities for sums of bounded random variables. In: The Collected Works of Wassily Hoeffding (1994)
18. Intel: Hyperledger Sawtooth description (2019), https://sawtooth.hyperledger.org/docs/core/releases/latest/architecture/poet.html, accessed: 2019-03-10
19. Kiayias, A., Russell, A., David, B., Oliynykov, R.: Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol. Cryptology ePrint Archive, Report 2016/889 (2016), https://eprint.iacr.org/2016/889
20. Kokoris-Kogias, E., Jovanovic, P., Gasser, L., Gailly, N., Syta, E., Ford, B.: Omniledger: A secure, scale-out, decentralized ledger via sharding. In: 2018 IEEE Symposium on Security and Privacy (SP) (2018)
21. Micali, S., Rabin, M.O., Vadhan, S.P.: Verifiable random functions. In: Proceedings of the IEEE Symposium on Foundations of Computer Science (1999)
22. Moran, T., Orlov, I.: Proofs of space-time and rational proofs of storage. In: Cryptology ePrint Archive, Report 2016/035 (2016)
23. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf (2008)
24. Rabin, M.O.: Transaction protection by beacons. Journal of Computer and System Sciences **27**(2), 256–267 (1983)
25. Syta, E., Jovanovic, P., Kogias, E.K., Gailly, N., Gasser, L., Khoffi, I., Fischer, M.J., Ford, B.: Scalable bias-resistant distributed randomness. In: 2017 IEEE Symposium on Security and Privacy (SP). pp. 444–460. Ieee (2017)